XP-001205334

♦ Optical Network Design and Restoration

Bharat T. Doshi, Subrahmanyam Dravida, P. Harshavardhana, Oded Hauser, and Yufei Wang

The explosion of data traffic and the availability of enormous bandwidth via dense wavelength division multiplexing (DWDM) and optical amplifier (OA) technologies make it important to study optical layer networking and restoration. This paper is concerned with fast distributed restoration and provisioning for generic mesh-based optical networks. We consider two problems of practical importance: determining the best restoration route for each wavelength demand, given the network topology and the capacities and primary routes of all demands, and determining primary and restoration routes for each wavelength demand to minimize network capacity and cost. The approach we propose for both problems is based on precomputing. For each problem, we describe specific algorithms used for computing routes. We also describe endpoint-based failure detection, message flows, and cross-connect actions for execution of fast restorations. Finally, we report test results for large carrier-scale networks that include both the computational performance of the optimization algorithms and the restoration speed obtained by simulation. Our results indicate that subsecond restoration, high capacity efficiency, and scalability can be achieved without fault isolation and with moderate processing. We also discuss methods for scaling algorithms to problems with very large numbers of demands. The wavelength routing and restoration algorithms, the failure detection, and the message exchange and activation architectures we propose are collectively known as WaveStar™ advanced routing platform.

Introduction

Recent advances in dense wavelength division multiplexing (DWDM) and optical amplifier (OA) technologies have created the potential for enormous bandwidth in a single optical fiber. Terabits/second bandwidth in a single fiber is a near-term possibility. Simultaneously, advances in cabling technology are already permitting up to 432 fibers in a single fiber cable. Most deployment of DWDM and OAs today is for increasing point-to-point bandwidth, and the true networking is done at higher layers—synchronous optical network (SONET)/synchronous digital hierarchy (SDH), asynchronous transfer mode (ATM), and Internet protocol (IP). However, wavelength add/drop multiplexers (WADMs), wavelength cross connects (WXCs) of different types, and fiber cross connects will allow true optical layer networking in the near future. These optical networks may provide efficient interconnection of SONET/SDH rings, extend to interconnect service nodes, carry service traffic directly without intervening SONET/SDH layers, and even extend to the endpoints to provide transparent networking. The introduction of optical layer networking creates the need for routing wavelength demands over optical networks. Equally important, this creates a need for restoration in the optical domain.

With ultra-high-capacity fiber cables and optical cross connects, any fiber cut or cross-connect failure will be catastrophic unless rapid restoration of service is an integral part of the network design and operations strategies. This restoration typically involves rerouting demands around failure on an alternate path. Until recently, the core transmission network based on plesiochronous digital hierarchy (PDH) used general mesh topology for interconnecting digital cross-connect systems (DCSs) by point-to-point fibers. DCSs were also used to implement restoration strategy by rerouting service demands around failure. Advances in automated restoration algorithms decreased the restoration time from several hours to a few minutes. SONET/SDH self-healing rings provided a major breakthrough by allowing subsecond restoration. However, interconnected SONET/SDH rings with point-to-point optical layers also resulted in large capacity penalties compared to traditional mesh-based PDH or SDH networking.

As an optical layer evolves from providing point-to-point connectivity to becoming a networking layer in any of the forms described above, part or all of the network may not be protected by SONET/SDH ring restoration. This is true of the optical subnets interconnecting SONET/SDH rings and the optical networks interconnecting service nodes. Furthermore, many factors make it attractive to carry some of the service traffic (for example, fast-growing IP backbone traffic) directly over an optical network without the intervening SONET/SDH layer. In that case, the entire network would need a new restoration strategy. Of course, restoration could be provided at either the service layer or the optical layer. The relative benefits of the two have been debated and will continue to be debated. That, however, is not the focus of this paper. We believe that some or all services will benefit from restoration at the core optical layer. While the restoration time at this layer need not be strictly smaller than that provided by SONET/SDH self-healing rings, it should be close enough to that benchmark. Services that can tolerate a much longer restoration time may be restored at the service layer if the capacity savings justify the complexity of managing two restoration procedures.

It is possible to design a simple restoration strategy for optical rings. On the other hand, mesh networks promise flexible use of fiber capacity. The challenge is to design a restoration strategy that allows subsecond restoration for carrier-scale mesh-based optical networks. This is the challenge we address in this paper. We develop an approach based on distributed precomputation of restoration routes that provides scalability and fast restoration. In addition, failure-independent end-to-end restoration adds to the restoration speed in cases for which fault isolation is time consuming. Along with the algorithms designed to make the best use of capacity, we describe message exchanges and the cross-connect actions needed in some detail. The approach we developed for restoration can easily be enhanced to include provisioning the primary routes for demands. Thus, the paper also describes algorithms for simultaneous computation of primary route (for provisioning) and restoration route (to be invoked for restoration). This approach to distributed provisioning and restoration will speed up provisioning and will pave the way for dynamic reconfigurability of the entire optical layer network.

While this work was motivated by the need for optical layer restoration (and provisioning), it can be

Panel 1. Abbreviations, Acronyms, and Terms

ATM—asynchronous transfer mode
BFS—breadth-first search
DCS—digital cross-connect system
DWDM—dense wavelength division multiplexing
FIFO—first in, first out
IP—Internet protocol
(IP)—integer program
NP—nondeterministic polynomially bounded
OA—optical amplifier
OC-48—optical carrier digital signal rate of 2.488 Gb/s in a SONET system
OC-192—optical carrier digital signal rate of 9.953 Gb/s in a SONET system
OC-768—optical carrier digital signal rate of 39.813 Gb/s in a TDM system
PDH—plesiochronous digital hierarchy
SDH—synchronous digital hierarchy
SONET—synchronous optical network
TDM—time division multiplexing
WADM—wavelength add/drop multiplexer
WDM—wavelength division multiplexer
WXC—wavelength cross connect

used at the SONET/SDH, ATM, or IP layer to create fast restoration and provisioning without the need for powerful centralized processing. When applied to the service layer, the number of demands to be restored may be very large. We will discuss the way in which the algorithms we describe can scale to such networks in a subsequent section of this paper.

Although the problem motivation and the outcome involve real networks and products, this paper focuses more on the core algorithmic concepts and the messaging structure needed to implement the algorithms. Many practical details have thus been omitted.

The next section of this paper gives an introduction to general mesh network restoration and a literature review. The following section provides an overview of optical networking with its special constraints on restoration and defines the two problems related to restoration and provisioning. The next three sections present solutions to these two problems. Subsequent sections discuss potential implementation architectures for fast activation of restoration paths, report our computational results, and consider scalability. The final section concludes with some observations about this study and remarks about future work.

Alternatives For General Mesh Network Restoration

In this section, we discuss restoration in a general mesh-based network. A restoration scheme is usually evaluated by four important performance criteria: restoration speed, restorability, capacity efficiency, and algorithm scalability.

Restoration speed can be defined as the elapsed time between the moment at which a failure happens and the time at which traffic is restored. *Restorability* refers to the types of failure a proposed scheme can handle—link, node, or multiple failures. The *scalability* criterion requires the time to compute the routes as well as the restoration time remains acceptable as the network size and the number of demands grow. *Capacity efficiency* can be measured by many parameters:

- The fraction of demands that can be restored given the network capacities, point-to-point demands, and primary route for each demand;
- The needed network capacities for 100% restoration given point-to-point demands and primary route for each demand; or
- The total network capacities for primary and restoration routes given point-to-point demands and 100% restorability.

As mentioned above, restoration speed is a key concern for optical networks.

Many restoration schemes have been proposed for generic mesh networks.[1] They can be classified by their route computation and execution mechanisms as centralized versus distributed, by their type of rerouting as link-based versus path-based, and by their computation timing as precomputed versus real time (after failure has occurred).

Link-based restoration methods[2,3] reroute disrupted traffic around the failed link, while path-based rerouting[4,5] replaces the whole path between the two endpoints of a demand. The link-based approach requires the ability to identify a failed link at both ends of that link. This method also makes restoration more difficult in the event of a node failure. Furthermore, it limits the choice of restoration paths and thus may use more capacity.[6,1]

The precomputed approach[4,7,8,9] calculates restoration paths before a failure happens, while the real-time approach[2,10] does so afterward. The former allows prior availability of reroute information to the nodes where actions need to be taken after the failure is detected and hence allows fast restoration. The latter requires time to compute the alternate route after failure is detected and hence is likely to be slower.

Centralized restoration methods[7,11,12,13,14] compute restoration (and primary) paths for all demands at a central controller where up-to-date information concerning the network is assumed to be available. The routes computed by this central controller may then be downloaded into nodal databases for primary and restoration route tables. These algorithms are generally path based. They may involve capacity discovery after the failure is detected or precomputation of routes. In the former case, alarm messages from affected network elements are used to identify failure, ascertain the remaining topology and capacity, and then find the best alternate routes for the affected demands. This is necessarily slow but can be very efficient from the perspective of capacity efficiency. However, given the importance of restoration speed and the potential difficulty in fast failure isolation in optical networks, this approach is not attractive. Thus, the centralized methods of interest to us are the ones involving precomputed routes.[7] Because of the total knowledge of topology and capacities at the central controller, these algorithms usually provide higher capacity utilization. However, maintaining up-to-date topology and capacity information at a central controller requires frequent communication between the nodal databases and the central controller. Moreover, centralized computation may not be scalable to large networks with large numbers of demands. This makes distributed approaches to computing restoration routes an attractive alternative for optical mesh networks.

Distributed methods may involve precomputed reroute tables[8,9] or real-time discovery of capacity and routes. Real-time capacity discovery-based methods[2,3,15,16,17] send out flooding messages that search for available link capacities after a link failure happens and then reroute the disrupted traffic around the failed link. The advantages of these link-based, real-time techniques are their simplicity and distributed nature. The disadvantages are their slowness, inefficient capacity utilization, and restorability limited to single-link failures (for example, node failures cannot be handled using approaches proposed in the literature.[2,3,15,16,17]). Thus, distributed precomputation of restoration route is attractive for the problem at hand.

Several distributed precomputed approaches have been proposed in the past.[5,8,9,10] In these strategies, restoration paths are failure dependent, which may allow better capacity utilization. However, a large number of restoration paths—one set for each failure scenario—must be precalculated and stored in memory. This creates a scalability issue. Moreover, potential difficulty in rapid failure isolation in optical networks makes this approach unattractive.

The restoration approach we propose is distributed, precomputed, path based, and failure independent. Thus, it overcomes all the disadvantages mentioned above. Failure-independent restoration is based on the disjoint path idea—that is, the service route and its restoration route are topologically diverse (node- and link-wise). Therefore, this scheme can restore any single link failure or node failure. Although every service route is assigned a restoration route, no dedicated capacity needs to be reserved for the restoration route, resulting in capacity savings. The disjoint path restoration and capacity savings idea was first proposed by Kawamura and Tokizawa[18] for an ATM virtual path setting and was further refined by Kawamura, Sato, and Tokizawa.[4] The restoration problem they addressed is the first one of the two problems we consider in this paper. In their approaches, restoration routes are either preprovisioned or precomputed by a simple centralized shortest-path algorithm, and capacity sharings are calculated by trying each possible failure, all in a centralized way. Our algorithm for this problem is not only fully distributed, but also has optimized capability to improve capacity utilization.

We choose the disjoint path restoration architecture because our main objectives are restoration speed and simplicity, although we do consider capacity optimization to be an important factor, which we address in several sections of the paper. To our best knowledge, no prior research has been done addressing capacity optimization of the disjoint path restoration. However, there is a considerable body of research in the literature that discusses optimization for link-based restorations[6,19,20,21] and failure-dependent path-based approaches,[6,22,23] all in a centralized way.

Problem Definition

This section provides an overview of optical networking with its special constraints on restoration and defines the two problems related to restoration and provisioning.

Optical Networking

An all-optical network[24,25] consists of fiber links connecting nodes equipped with wavelength division multiplexers (WDMs) and wavelength cross connects (WXCs). This network can provide pure optical transport between two network nodes at the wavelength level. WDMs effectively increase fiber bandwidth by multiplexing optical signals on several different wavelengths into a single fiber. WXCs function as wavelength switches.

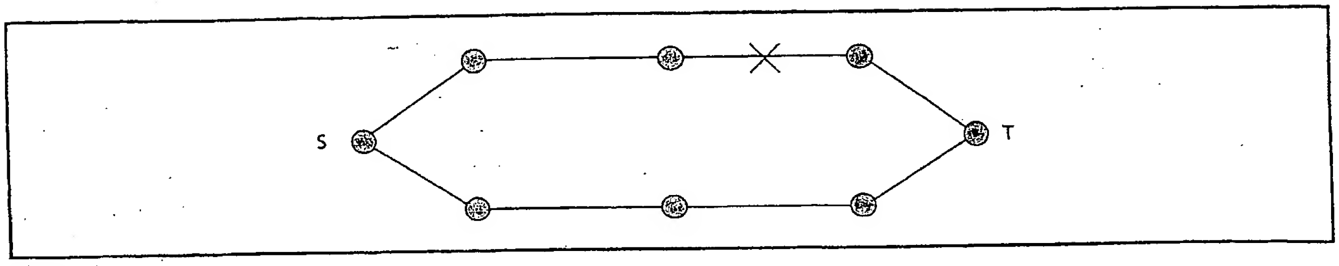


Figure 1.
Restoration architecture.

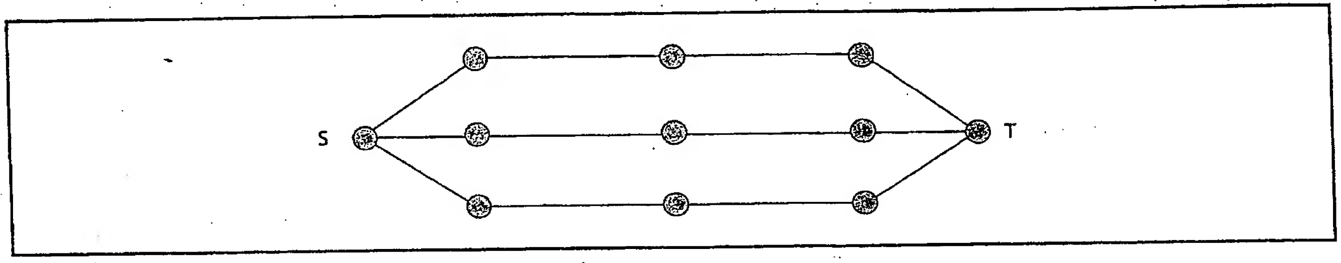


Figure 2.
Restoration route (alternate path) capacity sharing.

The WXCs may allow only wavelength-select function, or they may allow full wavelength-interchange function. A wavelength-select cross connect allows a signal on an incoming fiber to be switched to an outgoing fiber only on the same wavelength, whereas a wavelength-interchange cross connect allows a wavelength to be converted to another wavelength before switching the signal to another fiber. Many papers have been published discussing the capacity impact of the wavelength-interchange cross connect.[26,27] This is not the focus of the current paper; we assume the use of wavelength-interchange cross connects. However, the algorithms we describe can be extended to work with wavelength-select cross connects.

Restoration Architecture

We now describe the proposed restoration architecture for general mesh networks. Although we discuss this in the context of optical networks, the same architecture will work for SONET/SDH mesh, ATM mesh, or IP mesh networks. When dealing with the SONET/SDH or service-layer (ATM, IP) mesh networks, appropriate aggregation may be needed to achieve the desired scalability and speed. In addition, while this discussion focuses on restoration route computation, these methods can be extended to provide distributed provisioning as well as restoration.

A demand in an optical network is specified in terms of wavelength as the basic unit. For each demand, our approach uses a restoration route that is link and node disjoint from the service route (of course, the source node and the destination node are common for both routes). If the service route fails for any reason, its restoration route can be activated immediately, as illustrated in **Figure 1.** Obviously, no single node or link failure can bring down both the service route and the restoration route of a demand. Thus, our restoration strategy protects all single failures.

While seemingly similar, our restoration strategy is different from the 1 + 1 protection, which works at the link level, and the so-called 1 + 1 restoration (in which the source node bridges two copies of the signal—one along each route). In particular, the restoration route (the alternate path) in our strategy does not need dedicated capacity reservations. If the service (primary) routes of two demands are disjoint, no single failure will affect both demands simultaneously. This means that the restoration routes of these two demands can share link capacities, because these two routes will not be activated at the same time. **Figure 2** demonstrates this idea of restoration route capacity sharing by an example. In this figure, both demand d_1 and demand d_2 are between nodes S and T. Demand d_1's service route is the top path and demand d_2's service route is the bottom path. Both demands have the same restoration route, which is the middle path. Although the middle path is used by two demands, each link on the path needs only one unit of capacity because the two demands will never need restoration at the same time under the single-failure assumption.

Problem Definition

We now formally state the three basic problems related to restoration and provisioning.

Problem 1A: maximum restoration. *Given a network described by nodes (WXCs) and links, a set of point-to-point demands and their service routes in an optical network, and spare capacity on each link, find restoration paths for as many demands as possible under the link capacity constraints.*

Problem 1B: minimum capacity restoration. *Given the input in problem 1A, find restoration routes for all demands (100% restoration capability) so that the capacity requirement is minimized.*

Problem 2: joint optimization. *Given a network in terms of nodes and links as in problems 1A and 1B and a set of point-to-point demands, find both the service (primary) route and the restoration (alternate) route for each demand so that the total required network capacity is minimized.*

Problem 1 applies to the scenario in which the network is already carrying demands, and new demands are being added while some old demands are being removed. While it is not possible to change the primary (service) route for existing demands, the restoration routes—not in use in the absence of a failure—can be recomputed to get better capacity utilization. Problem 1A is the more realistic case in which we want to achieve the maximum possible restoration given the current network capacity. However, problem 1B addresses the short-term capacity-planning phase that provides guidelines on whether additional capacity will be needed in the near future based on current and expected demands. If a capacity shortage is detected in 1A (less than 100% restoration capability), it also provides an indication of capacity need. However, with appropriate planning, that should be a rare event. For obvious reasons, the algorithms used for 1A and 1B are almost identical (they are *dual*, in linear programming language).

Problem 2 is relevant in many situations, including those in which:

- Existing demands on a legacy network are transferred to an optical network,
- Periodic "cleaning" allows changes in service and restoration routes of all demands, and
- Long-term planning using projected demands involves estimating capacity needs for all the demands at a time point in the future.

Long-term capacity planning may use either distributed or centralized approaches; thus, both are of interest in this case.

The following details should be noted:

- While we focus on provisioning a large number of demands at one time, the algorithms we develop are easily adaptable to the more frequent case in which one or a few demands are provisioned at one time while existing demands are left untouched (at least as far as the service routes are concerned).
- The distributed computation of restoration routes, as defined in problem 1, can take place asynchronously of the demand change process. Thus, the distributed algorithms can periodically be invoked to recompute all restoration routes, leaving the service routes untouched. Any changes in the demand matrix are automatically taken into account by this recomputation. In this scenario, both service and restoration routes are computed in real time as the demands arrive (using a simplified version of the rules used in the algorithms we describe). Then, the periodic recomputation will optimize the restoration routes. Since the restoration routes are not used in the absence of failures, this recomputation can be quite frequent (more frequent than the rate of demand changes). Furthermore, the algorithm can run continuously in the background, responding to incremental changes in the network—for example, calculating restoration paths for newly provisioned demands.
- Depending on the time needed to repair a failure, we may want to recompute the restoration routes with the reduced topology after a failure. Invocation of the distributed restoration route computation will allow us to do that.

- In real capacity-constrained situations, we would like to prioritize the demands and ensure that the demands with high availability requirements are always assigned restoration routes. The algorithms described here can easily be modified to create the right search order so that these priorities can be honored. The details will be described in a future paper.

Although we have assumed the use of wavelength-interchange cross connects throughout this paper, it would be interesting to point out the complexity impact that the wavelength-select cross connect brings to these three problems. It is easy to observe that under the wavelength-select assumption, different wavelengths do not interact with each other at all for problem 1A. Therefore, the entire problem can be decomposed into a number of unrelated subproblems—one for each wavelength—that can be solved individually and independently. In each subproblem, link capacity is essentially defined as the number of fibers on a link. Problems 1B and 2 become more complicated under the wavelength-select assumption. The added dimension of complexity is the demand to wavelength assignment. The wavelength assignment problem has been well studied,[28,29,30] so we will not discuss it in detail.

Maximum Restoration Problem

First, we show that problem 1A is nondeterministic polynomially bounded complete (NP complete) by demonstrating that a special case is NP complete. More specifically, we claim that the disjoint path problem is a special case of problem 1A. The disjoint path problem is to find mutually node-disjoint paths between multiple source-destination node pairs in a network. This problem is known to be NP complete.[31] Let us consider all these source-destination node pairs as our demands, add an artificial node to the network, and connect the artificial node to all the nodes in the source-destination node pairs. Then, let us designate the two-hop paths through the artificial node between these source-destination node pairs as primary paths. Since all primary paths thus defined share the common fault, there is no capacity sharing for alternate paths. If we set the spare capacity to be 1 on every link, our restoration problem becomes the disjoint path problem. Therefore, problem 1 is also NP complete. Thus, for sufficiently large problems, even a centralized algorithm for problem 1 will require a heuristic approach. Distributed computation will certainly require one. We describe a distributed heuristic approach in this section.

As mentioned earlier, problems 1A and 1B can use the same basic algorithms. Problem 1B involves an additional step—finding the best locations to add new capacity to obtain restoration routes for unrouted demands with minimum additional capacity. Thus, we focus on problem 1A from this point on and refer to it as problem 1. Our approach to this problem uses distributed precomputation of restoration routes and allows progressively better capacity utilization. The basic idea of the algorithm is quite simple: The source node of each demand searches for its restoration route independently of other demands and, therefore, the algorithm is fully distributed and asynchronous. However, four basic issues must be addressed:

- How does the source node of a demand do its route search?
- If the source node of a demand wants to use a link on the restoration route of that demand, how can this source determine whether the link has sufficient spare capacity?
- How can deadlocks be prevented when multiple demands are simultaneously contending for link capacities during their searches (for restoration routes)?
- How can capacity utilization be optimized?

Accordingly, the algorithm has four basic procedures:

- Route (path) search,
- Link capacity control,
- Concurrent resource contention locking, and
- Capacity optimization.

These four procedures constitute the major components of our overall algorithmic procedure.

Algorithm Outline

By using these basic procedures as subroutines, we can now state the outline of the algorithm:

1. The source node of each demand executes the contention-locking procedure to lock out its contenders.
2. If the locking is unsuccessful, the source waits for a random amount of time and retries. Otherwise, it starts the procedure for searching for the restoration route.
3. Within the route search procedure, the source node initiates a distributed breadth-first search (BFS) algorithm to find a route from the source to the destination using only links with residual spare capacity. A link's residual spare capacity is determined by querying the link capacity control procedure.
4. If a route is found, the source stores it as the restoration route for the demand. Otherwise, it activates the optimization procedure, which tries to release some link capacities critical to the route construction of the current demand by changing the restoration routes of previously routed demands.

Algorithm Components

We now discuss the functionality and implementation of the four basic procedures outlined above. Note that there might be several different ways to implement each one.

Restoration route (alternate path) search procedure. Recall that we are given the demand set and the service route (primary path) for each demand. We are also given the remaining capacity on each link. The search procedure for a demand is to find a valid alternate path from the source to the destination. The validity of a path is defined by two criteria. First, it should only use nodes and links that are not on the primary path of the demand (except for the source node and the destination node). Second, it should only use links that have sufficient spare capacity, taking into consideration other contending demands already routed on the same links.

The procedure can be implemented by a distributed BFS over a residual network. The residual network is constructed from the original network by removing nodes and links of the primary path and by removing links with insufficient capacities after querying the link control procedure. In actual implementation, the removal of nodes and links does not have to be physical. The nodes and links of the primary path can be included in a "forbidden" list, which is passed from node to node as the BFS progresses. The links do not need to be queried for their residual capacities before BFS begins. As BFS progresses, whenever it visits a link, it first queries the link capacity control procedure and only uses the link if the link has sufficient capacity.

Link capacity control procedure. This procedure determines if a link has sufficient capacity to accommodate a demand's restoration route. In order to answer this question, the procedure must take into consideration the total spare capacity of the link, all the demands whose restoration routes currently use this link, the contention relationship among those demands, and the new demand. Two demands are said to be *contending* if their service routes share a common node or a common link. When they do, the two service routes may fail at the same time from a single fault. Hence, restoration routes may need to be activated at the same time, thus contending for the same capacities. Conceptually, a simple way to determine the restoration capacity requirement on a given link is to consider all possible failures one at a time, calculate the capacity needed, and take the maximum across all failures. The following procedure is a fully distributed method that accomplishes this precisely in a very efficient way.

This procedure is implemented by maintaining a capacity utilization table at each link (more precisely, at one of the link's two end nodes), as shown in **Table I.** The table has multiple rows and three columns. Each row represents a possible fault (such as a node failure or a link failure). The first column identifies the fault. The second column is the set of demands affected by the fault that currently use the link in their restoration routes. The third column is the residual capacity—that is, the total spare capacity minus the restoration capacity consumed by the set of demands in column 2. **Figure 3** shows a specific example of a network with four demands, their service routes, and their restoration routes. In this example, the capacity utilization table at link D-E is given by Table I.

Table I. Link capacity control table for link D-E.

Fault ID	Affected demands	Link capacity requirement
Node B	d_2	1
Node C	d_1, d_4	2
Node F	d_2	1
Node G	d_1	1
Node I	d_1	1
Link A-C	d_1, d_4	2
Link A-B	d_2	1
Link B-F	d_2	1
Link C-G	d_1, d_4	2
Link G-I	d_1	1
Link F-M	d_2	1
Link I-M	d_1	1

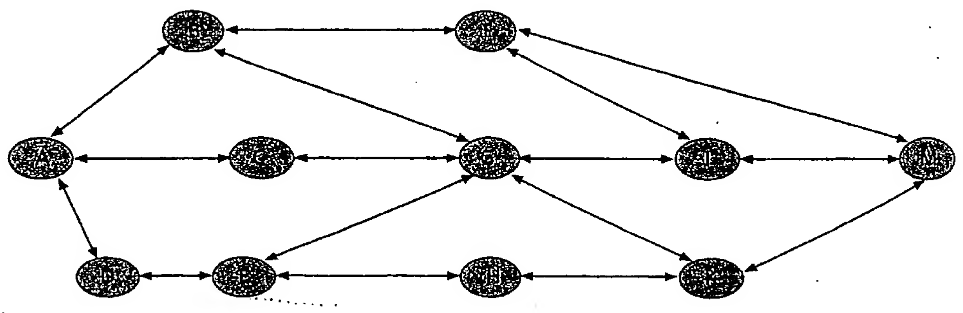
To answer a query from a new demand *d*, the link first determines *d*'s service route. Then it scans through all the rows of the table to see whether any faults affect *d*. If so, the link temporarily decreases its third column by one. It then calculates the minimum of all the entries in the third column plus one. A positive return value means yes—that is, it does have enough capacity to accommodate the new demand for its restoration route. A zero means no—it does not have enough spare capacity remaining. If the return value is zero, the link also determines *d*'s critical contenders—the intersection of all sets in column two of the rows that have the minimum values in column three.

Initially, the table is empty. Whenever a demand *d* wants to use the link on its restoration route, it sends the link its service route (nodes and links). Capacity is reserved by updating the table. Scan through the first column—if a fault affects *d*, add *d* into the set in the second column and decrease the value by one in the third column. If the existing rows do not cover all possible faults that can affect *d*, add additional faults by creating new rows, with a singleton set {*d*} in the second column and the total capacity minus one in the third column.

If, at any stage in the algorithm execution, the restoration route of a demand *d* no longer uses this link, capacity release is performed by updating the table. Scan through the first column. If a fault affects only *d*, delete the entire row. Scan through the second column. If a set contains *d*, delete *d* from that set and increase the value in the third column by one.

Concurrent contention-locking procedure. Although the algorithm may process multiple demands at the same time, no two of the demands can be contending; otherwise, there would be deadlocks during path searches. This problem is analogous to collisions on Ethernet. In fact, here the problem is even more complex because resource contention happens in both the time dimension and the space (network) dimension. Ethernet has contention problems only in the time dimension because of the common bus architecture. To avoid deadlocks, contention must be resolved before the path search begins. In this way, one demand at most out of a set of contending demands will be searching for a restoration route at a given time. Of course, noncontending demands can conduct their path searches at the same time. This nonconcurrent-contention condition is enforced by the contention-locking procedure, which is implemented as follows.

The basic idea is that a demand can inform the network of a lock along its service route in order to lock out contenders looking for restoration routes. Each node and each link has one and only one token. Before a demand starts its search for a restoration route, the source node of the demand sends out a "locking" message along its service route to the destination node. The message collects tokens from every node and link it traverses. It turns around and reverses the trip upon encountering the first node or link whose token is not available, and then it releases all the tokens it has collected on its way back to the source node. Upon arrival of the message (the demand did not win the locking), the source node immediately becomes silent for a random amount of time, then sends out the locking message again. If the message successfully reaches the destination node, the destination node knows that there is currently no other contending demand searching for a restoration route and that all the nodes and links on the demand's service route have been locked. The destination node can either start the route search procedure right away or notify the source node to initiate the search for the



Demand	Service route	Restoration route
d_1	A-C-G-I-M	A-D-E-H-K-M
d_2	A-B-F-M	A-D-E-H-K-M
d_3	B-F-M	B-G-I-M
d_4	A-C-G	A-D-E-G

Figure 3.
Capacity sharing.

restoration route for this demand. When either the source or the destination finishes its route search, it sends out an "unlocking" message to the other end along the service route. The message releases a token back to every node and link it traverses along the service route. Without this unlocking step, it would be impossible for another contending demand to establish a lock and begin its own search.

Optimization procedure. Because of the distributed nature and lack of global coordination of the route search procedure, the resulting routes may need more capacity than a centralized algorithm is able to achieve. This means that more demands might obtain their restoration routes (with the same link capacities) than the above route search procedure can handle. The optimization procedure is designed to improve capacity utilization by finding restoration routes for more demands. Whenever a demand fails to find a restoration route in the above route search procedure, it activates the optimization procedure, which tries to construct a restoration route for itself by changing the restoration routes of previously routed demands. This optimization procedure is described in detail in Appendix A.

Joint Optimization: Centralized Computation

In this section, we consider algorithmic approaches for problem 2 as defined in the "Problem Definition" section. Unlike the case of problem 1, both centralized and distributed computation may be of interest here. We consider the centralized case first. We begin with a more precise mathematical description of the problem, followed by an integer programming formulation. We then present two heuristic algorithms for problem 2. The first algorithm, labeled algorithm G, is based on a Lagrangean relaxation of the integer programming formulation. The second algorithm, labeled algorithm L, is based on local rerouting and is expected to be a faster alternative.

Problem Formulation

First, we need to introduce some notation. We view the network as a directed graph. Let N be the set of nodes, $L \subset N \times N$ be the set of links, and $D \subset N \times N$ be the set of demands. For each link $(ij) \in L$, let c_{ij} (a positive real number) be the capacity weight for link (ij), which can be regarded as a measure of capacity consumption per wavelength on the link. These weights are used to differentiate links from the capacity cost point of view—for example, by link distance. For a given design solution, let W_{ij} be the resulting capacity requirement on link (ij) in terms of number of wavelengths. Then, the objective function we want to minimize is $\sum_{(ij)\in L} C_{ij}W_{ij}$.

If $C_{ij}=1$ for all the links, then the objective is to minimize the total number of wavelength links. If C_{ij} is set equal to the mileage of the link, then the objective is to minimize the total wavelength miles.

We would like to point out that capacity optimization as defined above might not always be the most desired objective. For example, in the case of "green-field" network design, one may be interested in minimizing total network infrastructure cost, which consists of link cost and node cost. Link cost includes the cost of laying (leasing) the fibers. Node cost includes the cost of nodal equipment such as multiplexers, switches, and ports. While port and link costs can easily be included in the above model, the fixed cost of nodal equipment is not directly reflected there. However, we have been able to relate the various cost elements in a way that allows a two-parameter characterization of the overall cost structure. These two parameters are the per-wavelength cost and the per-fiber cost, and the above model can reflect these.

Next, we give the integer programming formulation. Let F be the set of all possible (single) faults. Since a fault can be either a failed node or a failed link, $F=N\cup L$. For each $f\in F$, let L_f be the set of links affected by fault f. More specifically, if f is a link fault, then L_f contains only the failed link. If f is a node fault, however, L_f contains the set of links subtending to the failed node—that is, $L_f=\{(ij)\in L: j=f\}$. For each $d\in D$, let s_d and t_d be the source node and the destination node of demand d, respectively. We need to define the following variables:

X_{ij}^d : binary,

1 if demand d's primary path traverses link (ij),

0 otherwise.

Y_{ij}^d : binary,

1 if demand d's alternate path traverses link (ij),

0 otherwise.

Z_{ijf}^d : binary,

1 if demand d is rerouted through link (ij) under fault f,

0 otherwise.

W_{ij} : nonnegative integer,

total number of wavelengths required on link (ij).

Problem 2 can be formulated as the following integer program (IP):

$$\min \sum_{(ij)\in L} C_{ij}W_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j:(ij)\in L} X_{ij}^d - \sum_{j:(ji)\in L} X_{ji}^d = \begin{cases} 1, & i=s_d \\ -1, & i=t_d, \\ 0, & \text{otherwise}\end{cases} \quad d\in D \tag{2}$$

$$\text{(IP)} \quad \sum_{j:(ij)\in L} Y_{ij}^d - \sum_{j:(ji)\in L} Y_{ji}^d = \begin{cases} 1, & i=s_d \\ -1, & i=t_d, \\ 0, & \text{otherwise}\end{cases} \quad d\in D \tag{3}$$

$$\sum_{j:(ij)\in L,(ij)\notin L_f} Z_{ijf}^d - \sum_{j:(ji)\in L,(ji)\notin L_f} Z_{jif}^d = \begin{cases} \sum_{(kl)\in L_f} X_{kl}^d, & i=s_d \\ -\sum_{(kl)\in L_f} X_{kl}^d, & i=t_d, \\ 0, & \text{otherwise}\end{cases} \quad \begin{matrix} d\in D \\ f\in F\end{matrix} \tag{4}$$

$$Z_{ijf}^d \le Y_{ij}^d, \qquad d\in D, (ij)\in L, f\in F, (ij)\notin L_f \tag{5}$$

$$\sum_{d\in D} X_{ij}^d + \sum_{d\in D} Z_{ijf}^d \le W_{ij}, \qquad (ij)\in L, f\in F, (ij)\notin L_f \tag{6}$$

The objective function (1) is the total weighted capacity requirement. Constraints (2) and (3) are the flow conservation constraints for demand d's service route and restoration route, respectively. Constraint (4) enforces the logical relationship whereby the restoration route consumes link capacity if and only if the service route is affected by the fault in question. Constraint (5) ensures that the restoration route of a demand is independent of the failure. Constraint (6) determines the link capacity requirement. Note that there is no explicit constraint to guarantee the disjointness of service and restoration routes. The disjointness is implied through Z variables. Further, note that although it appears that node faults always cover link faults because a link fails if either of its two end nodes fails, there is one exception. If there is only one link on the service route (one hop), then no node fault can cover the impact of the link fault. We implicitly assume that a demand does not need to be restored if either of its source or destination nodes fails.

Algorithm G

It is easy to observe that in the integer program (IP), constraint (6) is the only bundling constraint that ties different demands together. Without constraint (6), the problem would decompose to a number of independent subproblems—one for each demand. This decomposition would significantly reduce the computational complexity of the original problem. A common technique used to relax such constraints is the Lagrangean relaxation method.[32] The relaxation simplifies the problem, but the penalty is that the relaxed problem can only provide a lower bound to the original problem. Moreover, a solution of the relaxed problem may not necessarily be a feasible solution to the original problem. We take the Lagrangean relaxation approach by relaxing constraint (6). In our case, we can always make a solution of the relaxed problem into a feasible solution to the original problem, because constraint (6) can always be satisfied by appropriately selecting the W_{ij} variables.

To remove constraint (6), we need to introduce the Lagrangean multiplier $\lambda_{ijf} \geq 0$ and add the penalty term $\sum_{(ij)\in L, f\in F, (ij)\notin L_f} \lambda_{ijf}\left(\sum_{d\in D}(X_{ij}^d + Z_{ijf}^d) - W_{ij}\right)$ to the objective function (1). With some algebraic manipulation, one can easily see that the relaxed problem is meaningful only if we add the constraint $\sum_{f\in F, (ij)\notin L_f} \lambda_{ijf} = C_{ij}$ to the Lagrangean multipliers in addition to their nonnegativity constraints. The original problem is decomposed into a series of subproblems, one for each demand. After some algebraic transformations, for each demand d, we arrive at the subproblem in the following form:

$$\min \sum_{(ij)\in P_d} C_{ij} + \sum_{(ij)\in A_d} \sum_{f\in F_{Pd}} \lambda_{ijf}$$

such that

(IP$_d$) P_d is a path (route) connecting s_d and t_d

A_d is a path (route) connecting s_d and t_d

P_d and A_d are node disjoint except at s_d and t_d

where F_p is the set of all possible faults that can bring down route P.

Even though there is only one demand involved in problem (IP$_d$), the capacity sharing between the restoration routes of different demands is reflected by the Lagrangean multipliers. Since $\sum_{f\in F, (ij)\notin L_f} \lambda_{ijf} = C_{ij}$, the link cost C_{ij} for the restoration routes can be viewed as being split across different faults. For each link (ij), the fraction of C_{ij} charged to a restoration route for using the link depends on which faults affect the corresponding service route. More specifically, λ_{ijf} can be interpreted as the cost charged to a restoration route if this route uses link (ij) and the service route of the demand contains fault f. To establish the relationship between (IP$_d$) and (IP), we provide the following theorem.

Theorem. For each set of feasible values of λ_{ijf}, let v_d be the optimum value of (IP$_d$)—the objective function value of its optimal solution—for each $d \in D$, and let v be the optimum value of (IP). Then, $\sum_{d\in D} v_d \leq v$.

The proof, provided in Appendix B, is needed because this decomposition is not a standard Lagrangean relaxation. The subproblem (IP$_d$) is substantially different from its decomposed forms.

For each fixed set of λ values, let $v(\lambda) = \sum_{d\in D} v_d$, the optimum value of the relaxed problem. $v(\lambda)$ provides a lower bound to the original problem. In order to get a good lower bound, we attempt to adjust the λ

values in some way so that $v(\lambda)$ can be increased. Since $v(\lambda)$ is a piece-wise linear function of the values of λ_{ijf}, we must use a λ value adjustment procedure called *subgradient optimization*. It can be verified that the subgradient of $v(\lambda)$ as a function of λ is an array

$$\nabla v(\lambda) = \left[a_{ijf}: \quad (ij) \in L, f \in F, (ij) \notin L_f\right],$$

where

$$a_{ijf} = \sum_{d \in D} (X_{ij}^d + Z_{ijf}^d) = \sum_{d \in D:\, (ij) \in P_d} 1 + \sum_{d \in D:\, (ij) \in A_d, f \in F_{P_d}} 1 .$$

Because of the constraint $\sum_{f \in F,\, (ij) \notin L_f} \lambda_{ijf} = C_{ij}$, we need to use the projected subgradient $\nabla_p v(\lambda) = \left[a_{ijf} - \frac{\sum_{f \in F:\, (kl) \notin L_f} a_{klf}}{m-3} \right]$,

where m is the total number of nodes and links and $m-3$ is the total number of faults that does not affect a given link.

Iteration on λ values by subgradient optimization to improve the lower bound is the basic idea of algorithm G. At each iteration, the solutions of all the subproblems constitute a feasible solution to the original problem (IP) although, theoretically, nothing can be concluded about the feasible solutions obtained this way. An ascending sequence of lower bounds can indicate the merits of those feasible solutions. The algorithm terminates when either the duality gap (the difference between the feasible solution and the lower bound) is small enough or the improvement of the lower bound is too small. The best feasible solution at this point is chosen as the final solution.

Next, we discuss how each subproblem can be solved. One would expect that subproblem (IP$_d$) is easily solvable. Unfortunately, (IP$_d$) itself is an NP complete problem. It is an easy exercise to reduce the problem of integral flow with bundles, which is NP complete,[31] to (IP$_d$). Nevertheless, (IP$_d$) is substantially simpler than the original problem (IP). We propose the following algorithm R to solve subproblem (IP$_d$). The algorithm can either solve the subproblem to optimality or generate a good feasible solution. Recall that we can only construct a lower bound for (IP) if we solve each subproblem (IP$_d$) to optimality. But in case the algorithm can only find a feasible solution to (IP$_d$), we can use a lower bound of (IP$_d$) instead of its optimal solution to construct a lower bound for (IP). Therefore, we also provide an effective way to generate a good lower bound of the subproblem in case of failure to solve it optimally.

Algorithm R. The outline of algorithm R is as follows:

1. Find the shortest node-disjoint pair of paths between s_d and t_d as the initial solution (see Suurballe's algorithm[33]). Take the shorter path, P_d, as the service route and the longer one, A_d, as the restoration route. Break the tie arbitrarily. Calculate the objective value $v(P_d, A_d)$ of (IP$_d$). (P_d, A_d) is called the best feasible solution and $v(P_d, A_d)$, the best value.
2. Compute a set of K shortest paths between s_d and t_d in ascending order of path cost, where K is a parameter. We can stop early if either the current path cost is already no less than the best value thus far or K paths do not exist.
3. For each path P in the path set, choose it as a service route. With P fixed, compute the best restoration route A as the shortest disjoint path (from P) using the new link cost metric: for link (ij), $R'_{ij} = \sum_{f \in F_{P_d}} \lambda_{ijf}$. A may not exist. If it does, compute the objective value $v(P, A)$. If it is less than the best value, replace the best value and the best feasible solution by (P, A) and $v(P, A)$, respectively.
4. If the path cost is already no less than the best value for some path in the path set, stop. The current best feasible solution is optimal. If this never happens, use the best feasible solution as the final solution. In this case, use the cost of the last path in the path set as a lower bound.

Note that the solution produced by algorithm R is always optimum if the value of K chosen is large enough. However, we still provide the feasible solution as a fallback alternative just in case the K shortest-path computation is too time consuming for large networks with large K. After all the subproblems are solved, a new lower bound for (IP) is constructed as the sum of either the optimal value of (IP_d) or its lower bound.

Let S be a feasible solution of (IP). Let $C(S)$ be the objective function value (total capacity requirement) of S. $C(S)$ can be efficiently calculated by using the procedure discussed in the "Link Capacity Control Procedure" section. Now we are ready to state algorithm G.

Algorithm G. The outline of algorithm G is as follows:

1. Initially, set all λ values equally and scale them properly so that $\sum_{f\in F,\,(ij)\in L_f}\lambda_{ijf} = C_{ij}$. For each demand $d \in D$, call algorithm R to solve subproblem (IP_d). All the subproblem solutions collectively become an initial feasible solution S_0 to the original problem (IP). Set the best feasible solution (thus far) $S^* = S_0$.
2. At the k^{th} iteration, update λ values as $\left[\lambda_{ijf}^{k+1}\right] = \left[\lambda_{ijf}^{k}\right] + \alpha\nabla_p v\left(\left[\lambda_{ijf}^{k}\right]\right)$, where α is the step size. α is chosen so that λ^{k+1} is still nonnegative and adaptive to the improvement of the last iteration. Solve all subproblems (IP_d) and obtain a feasible solution S_{k+1}. If $C(S^*) > C(S_{k+1})$, set $S^* = S_{k+1}$.
3. Keep repeating step 2 until either $v(\lambda)$ cannot be improved or the value of $C(S^*)$ does not decrease for a predefined number of iterations.
4. S^*, the best feasible solution obtained thus far, can be further improved by a local rerouting procedure (such as algorithm L, which we discuss in the next subsection).

Algorithm L

Algorithm L is a local improvement procedure. The basic idea is to start with an initial solution. Then improve the solution by modifying the service and restoration routes of demands one at a time.

The outline of algorithm L is as follows:

1. For each demand $d \in D$, find the shortest node-disjoint pair of paths between s_d and t_d as the initial solution. Calculate the capacity requirement on each link.
2. For each demand $d \in D$, with its primary path (service route) P_d fixed, modify its restoration route, A_d, as follows. First, remove A_d and update link capacities. Second, define a link cost metric: For each link (ij), run the link capacity control procedure to check whether its capacity increases if it is used by d's restoration route. If so, $R_{ij} = C_{ij}$. If not, $R_{ij} = 0$. Set $R_{ij} = \infty$ for all the links incident to P_d to ensure disjointness. Third, take the shortest path between s_d and t_d under the new cost metric as the new restoration route for d.
3. For each demand $d \in D$, with its restoration route, A_d, fixed, reroute its service route, P_d, as follows. First, take out P_d and update link capacities. Although A_d remains unchanged, in the absence of P_d, it does not contribute toward capacity requirement at any link on its route. Second, define link cost metric: For each link (ij), run the link capacity control procedure to check each link (ij) on A_d and determine whether its capacity increases if the three faults i, j, and (ij) are introduced to d's service route. Let L_{ij} be the set of links on A_d whose capacity increases. Define $R_{ij} = C_{ij} + \sum_{(kl)\in L_{ij}} C_{kl}$. Set $R_{ij} = \infty$ for all the links incident to A_d to ensure disjointness. Third, calculate the shortest path between s_d and t_d under the new cost metric. If the total cost is improved, take this new parh as the new service route for d.

4. Repeat steps 2 and 3 in combination or in any order, and stop when no further improvement is possible. Note that different ordering of demands for rerouting may result in different improvements.

Because this algorithm is a local heuristic procedure, its results are sensitive to the initial solution and to the ordering of demands. However, the advantages of this algorithm are that it is fast, simple, and easy to accommodate additional constraints. Of equal importance is the fact that it is relatively easier to create a distributed version of algorithm L, which is the subject of the next section. Later, we will evaluate the optimization performance of algorithms L and G as centralized algorithms.

Joint Optimization: Distributed Computation

We now briefly describe a distributed version of algorithm L. In simple terms, each node conducts its computation independently of others. Obviously, if there is no capacity sharing, different demands do not interact with each other and the algorithm is perfectly distributed. With capacity sharing, some kind of coordination between demands is necessary.

In order to make algorithm L distributed, let us examine the three steps which compose the algorithm. For step 1, since each demand can do a shortest disjoint path-pair computation independently of other demands, the step is fully distributed. Step 2 reroutes alternate paths with primary paths fixed. This step is the uncapacitated version of the path search procedure of the distributed algorithm for problem 1. It essentially attempts to reroute the restoration route for each demand by using free capacity as much as possible. By *free capacity*, we mean that the use of a link by a restoration path does not increase capacity requirement according to the link capacity control table. In order for queries to link capacity control tables to yield meaningful results, some locking mechanism is necessary. If we apply the contention-locking procedure of problem 1, two demands rerouting at the same time must have disjoint fault sets. This ensures that the answers each one receives from the link capacity control table at each link will be consistent, no matter which one queries first. Obviously, here we assume that on each link the maximum capacity required by any single fault stays constant for the duration of a demand's rerouting. This may not necessarily be true, but the changes should be minimal if each demand computes its route very fast. Therefore, step 2 becomes distributed by incorporating the contention-locking procedure. Step 3 reroutes service paths with restoration paths fixed. Similarly, two demands rerouting at the same time may conflict in their link capacity queries of their restoration routes due to changes in the fault set of their service routes. In order to avoid the conflict, we need the modified contention-locking procedure of problem 1 in which a demand locks its (fixed) restoration path before it starts to search for its (new) service path. This locking mechanism ensures that, at any one time, all demands doing service path computation have disjoint restoration paths. This condition is required for the shortest path metric in step 3 to be effective in reducing total capacity requirement. Since there is no capacity constraint for the joint optimization problem, there is no capacity contention among service path searches.

Note that steps 2 and 3 cannot be executed at the same time (for two different demands). Therefore, some global synchronization is necessary. We propose three phases, which are executed sequentially:

- *Phase 1.* All demands execute step 1 in parallel.
- *Phase 2.* All demands execute step 2 asynchronously.
- *Phase 3.* All demands execute step 3 asynchronously.

The synchronization between phases can be achieved by, for example, broadcasting or centralized locking. The centralized locking mechanism can be implemented as follows: Select a node in the network—for example, node A—as the locking host. During each phase, every demand turns up a flag (labeled by its demand ID) at node A before it starts its computation, indicating that the demand is engaged in computing. After computation is complete, the demand turns down its flag at node A. If any demand is finished with the current phase, before it can proceed to the next phase, it must check flags at node A. If there is an "up" flag, it must wait, because other demands have not finished with the current phase.

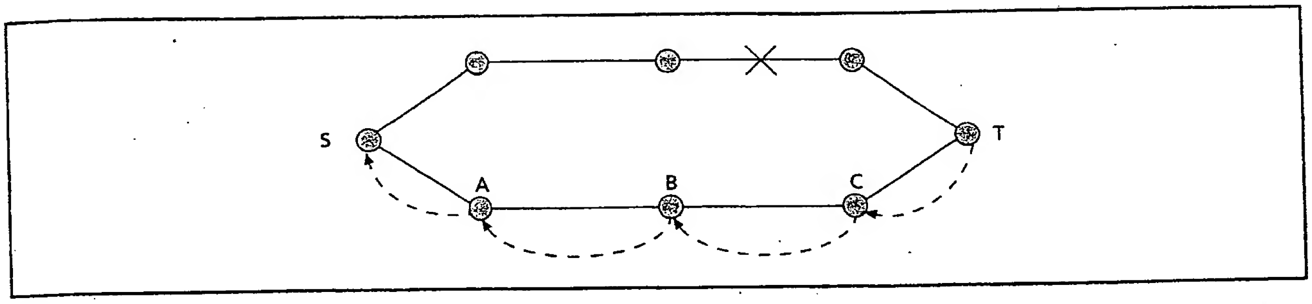


Figure 4.
Sequential architecture.

The demand will stay silent for a random amount of time and check node A again. It only starts its next phase of computation when there is no "up" flag at node A.

Restoration Activation Architectures

We now discuss restoration activation architectures. Since the restoration route for each demand is precomputed, information about this route can be stored at the destination node of the demand. Note that we have assumed unidirectional demand in our formulation. In reality, each end will be the destination for bidirectional demands and will have the restoration route stored. Because no reserved capacity is allocated for the restoration route, the route cannot be set up beforehand. The actual route setup (the establishment of switching mechanisms at cross connects along the route) can only be performed in real time after a failure occurs. Conceptually, the activation procedure should work as follows. Once the destination node detects failure of the service route, it first initiates cross-connect operations at every node on the restoration route to connect the path, then signals the source node to bridge the traffic to this route, and finally switches the signal selector for the demand to select the restoration route.

At a more detailed level, the activation procedure requires the following functions, which must be supported by the hardware: failure detection at the destination, internode message exchanges, message processing, cross-connect execution, traffic bridging, and signal selection. Before we proceed to the activation architectures, we assume that these functions are supported by the network hardware and software architectures.

We propose two ways to implement the activation procedure—one is a sequential architecture and the other is a parallel architecture.

Sequential Activation Architecture

Figure 4 illustrates the basic idea of the sequential activation architecture. The upper S-T path is the service route and the lower S-T path is the restoration route. The dashed lines represent flows of restoration activation messages. The destination node initiates the activation of the restoration route, and the route is connected by one node at a time starting at the destination node.

More specifically, the procedure works as follows. Upon detecting that the service route of a demand has failed, the destination node T first determines an incoming port number for the restoration route and switches its signal selector to receive a signal on this port. It then determines the corresponding outgoing port number at its preceding neighbor node C on the restoration route. Finally, it sends a cross-connect request message to node C. Node T encloses in the message two information items. The first one is the outgoing port number for the cross connection to be made at node C. The second is the restoration route topology together with demand ID. Upon receiving the message, node C determines the incoming port number and performs the cross connection. It then, in turn, sends a message to its preceding neighbor node B. The content of the message is similar—the outgoing port number at node B and the demand's restoration route topology. Upon receiving the message, node B will take the same action and send a message to the next node. This step is repeated, one node at a time, until the source node S is reached. Once node S receives the

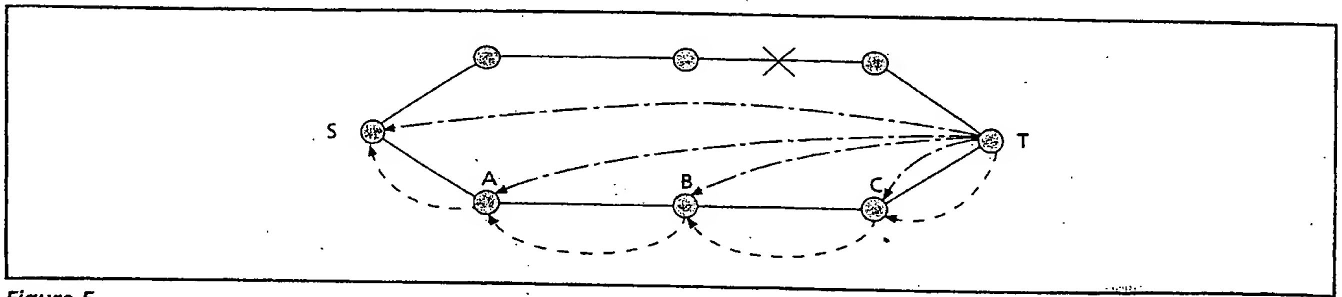


Figure 5.
Parallel architecture.

message, it first identifies itself as the source node and identifies the demand by the demand ID. It then bridges the signal of the given demand to the outgoing port whose number is obtained from the message.

This architecture is simple and straightforward. A significant shortcoming, however, is the slowness due to its sequential execution, especially for restoration routes with many hops. This is the reason to consider the parallel architecture that we discuss in the next section.

Parallel Activation Architecture

In order to speed up the connection setup of the restoration routes, we propose another mechanism that allows parallel operation of cross connects at different nodes. The penalty for achieving the parallelism is the extra messaging to communicate port assignments. Unlike the sequential architecture, there are two types of messages in this case. Messages of type 1 are messages that go from the destination node to all other nodes on the restoration route. Messages of type 2 travel between neighboring nodes on the restoration route. The function of these messages is discussed below. **Figure 5** illustrates the message flows. The upper S-T path is the service route and the lower S-T path is the restoration route. The dotted links represent type 1 message flows and the dashed lines represent type 2 message flows.

The procedure works as follows. Upon detecting that the service route of a demand has failed, the destination node T first retrieves the restoration route of the affected demands and sends out a type 1 message to every node on the restoration route. Each type 1 message contains three information items: the message type, the demand ID, and the node ID of the node immediately preceding the receiving node on the restoration route. Node T then determines an incoming port number for the alternate path and switches its signal selector to receive a signal on this port. Finally, it determines the corresponding outgoing port number at its preceding neighbor node C on the restoration route and sends a type 2 message to node C. Each type 2 message contains three information items as well: message type, demand ID, and the outgoing port number for the cross connection to be made at the receiving node. After receiving a type 1 message, each intermediate node takes the following actions:

1. It determines an incoming port number for its cross-connect operation and determines the corresponding outgoing port number at its preceding neighbor node.
2. It then sends a type 2 message to its preceding neighbor node.
3. After receiving both a type 1 message and a type 2 message, each intermediate node executes the cross connection.
4. Once the source node S receives a type 2 message from node A, it identifies the demand by the demand ID and bridges the signal of the demand to the outgoing port whose number is obtained from the message.

Further Improvements on Restoration Speed

There are other possibilities for accelerating failure detection and activation of restoration routes. These include reduction of message processing and exchanges, reduction of cross-connect activations, and dedication of signaling channels. This subsection gives a brief description of these strategies.

Message processing and exchange reduction. Our approach to reduce nodal message processing and internode message exchange is to preassign channel numbers to the restoration routes of the demands on a link-by-link basis. In this context, *preassignment* means that channel assignment is done before a failure occurs. Note that such an assignment is failure independent, even though multiple demands may be assigned to the same channel as long as they are not contending (that is, they do not require simultaneous activations of their restoration routes under any single fault scenario). Channel preassignment on a link is clearly doable if every restoration route has dedicated capacity reserved on the link. In the case of capacity sharing, channel preassignment sometimes may require extra link capacity, as illustrated by the following example.

Assume that there are three demands: d_1, d_2, and d_3. All of their restoration routes use a given link *l*. Let us further assume that the capacity control table of link *l* is given as **Table II.**

From the table, we can see that d_1, d_2, and d_3 are pairwise contending, but no single fault can bring down all three demands. The link capacity required is two units. But within the capacity of two units, there are only two channels. If there were a feasible way to preassign two channels to three demands, at least two of the demands would be assigned to the same channel—for example, d_1 and d_2 would be assigned to the same channel. Since f_1 could bring down both d_1 and d_2, they would contend over the same channel at the same time—a contradiction. Therefore, in order to do channel preassignment on link *l*, we need three units of capacity. Note that, although three also happens to be the capacity required by the restoration routes without capacity sharing in this example, channel preassignment capacity is usually much less. In fact, based on past experience with similar problems in SONET/SDH rings, we expect that the percentage penalty of channel preassignment will decrease rapidly as the number of demands per link increases. In fact, channel preassignment can be formulated mathematically as a graph-coloring problem, in which nodes represent demands and a link exists between two nodes if the two demands they represent are contending. We developed a fast heuristic algorithm for the graph-coloring problem and conducted computational tests to assess capacity penalty on channel preassignment.

Table II. Example for channel preassignment.

Fault ID	Affected demands	Link capacity requirement
f_1	d_1, d_2	2
f_2	d_2, d_3	2
f_3	d_3, d_1	2

Our observation is that, most of the time, no extra capacity is needed for most links. Even for links that require additional capacity for channel preassignment, the capacity increase required is insignificant.

Once channels are preassigned to demands, there is no need to communicate port numbers between nodes at the time of restoration activation. The only information an intermediate node needs is the IDs of failed demands before it executes cross-connect operations, because input-output port mappings are predetermined and stored in local memory.

Cross-connect reduction. One of the insights we gained from our simulation studies is that the bottleneck for restoration speed is the large number of cross-connect requests imposed, usually at a small number of nodes (a single node in many cases). Therefore, an effective way to speed up restoration is to somehow release some load from the maximally loaded (critical) nodes. One approach is to perform cross connection prewiring at these critical nodes. The prewiring made at each node is independent of individual demands. The prewiring can be viewed as a virtual express link that bypasses the node.

More specifically, for each critical node, find all the demands whose restoration routes go through the node. Then, for all the links incident to the node, examine all possible pairwise link combinations. For each such pair of links l_1 and l_2, find the minimum restoration traffic flow that traverses both l_1 and l_2 under all fault scenarios. If the minimum flow is *k* units, then we can prewire *k* channels from link l_1 to link l_2. The cross connections prewired this way will not need to be disassembled under any fault scenario. At the time of activation, *k* demands are effectively removed from this node's cross-connect load.

Dedicated signaling channels. Another approach we have considered for restoration speedup is to provide a dedicated signaling channel for each demand

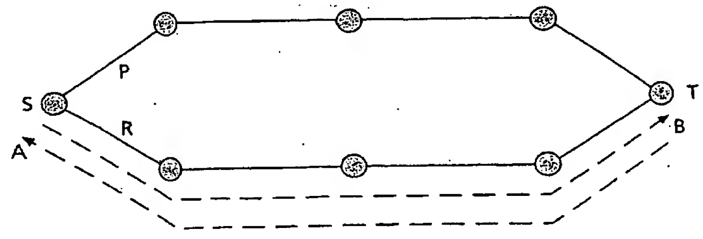


Figure 6.
Dedicated signaling channel per demand.

for faster messaging. The signaling channel can speed up message exchanges in two ways. First, this channel can be provisioned at a lower layer to bypass the protocol stack and to avoid significant software processing. Second, the messages travel faster using the dedicated channel. **Figure 6** illustrates the basic idea. For a given demand between node S and node T, the top path is its service route, marked "P," and the bottom path is its restoration route, marked "R." The two dashed lines, marked "A" and "B," are the two dedicated signaling channels for this demand along its restoration route—one per direction of transmission. Once node T detects failure of the service route, it sends out restoration messages through signaling channel B to all the intermediate nodes on the restoration route and to node S. Since each signaling channel is dedicated to an individual demand, the communication delay should be minimal in the absence of resource contention and queuing delay. In addition, these signaling channels do not need their own survivability mechanisms. Those are only needed when their service routes fail. Under the single failure assumption, they should never fail when they are needed.

Test Results

We next report the results of a number of preliminary tests we conducted on both small sample networks and large nationwide networks. The tests focused on two evaluation criteria: restoration speed and optimization performance. Solutions to both problem 1 and problem 2 were tested. For these examples, we made very conservative assumptions about the processing speeds available to execute restoration. In particular, the processing power assumed corresponds to that in today's cross connects, as opposed to that in voice and ATM switches. Restoration can be much faster with faster processors.

Restoration Speed

We developed a simulation model to test the speed of our restoration architecture. This simulation was run with rather conservative assumptions regarding the processor and hardware performance associated with WXCs and clients at two ends. The assumptions are:

- *Internode messaging* is accomplished through network supervisory channels. Message travel time (propagation delay) between two nodes is 8 ms per 1,000 miles of the distance between the two nodes.
- *Node message processing* is performed at a speed of 20 ms per message. Each node processes messages (either incoming or outgoing), one at a time. When a node processor is busy, messages are buffered in a first-in–first-out (FIFO) queue.
- *Cross-connect instruction* can be executed in 10 ms. Multiple cross-connect instructions can be bundled and executed in parallel. A cross-connect request must be processed as a message before the cross-connect instruction can be generated.
- *Traffic bridging and signal selector switching* is accomplished in 10 ms per operation—the same as for cross-connect operations.
- *Failure detection* requires 5 ms per detection.

The restoration activation architecture simulated is the parallel architecture. We use three test examples. One is a sparse nationwide network and the other two are larger, denser nationwide networks. **Table III**

Table III. Network and traffic sizes.

Network	Number of nodes	Number of links	Number of demands (wavelengths)
A	28	48	50
B	70	103	156
C	301	449	372

Table IV. Restoration time.

Network	Maximum XC load (number of activations)	Restoration time
A	4	130 ms
B	12	505 ms
C	13	572 ms

XC – Cross connect

Table V. Number of unrouted demands.

Network	Distributed solution	Optimal solution
1	16	14
2	12	9
3	10	9
4	14	12
5	9	9
6	11	11
7	8	8
8	9	9
9	8	8
10	10	9

introduces these networks and their sizes. Note that each wavelength may represent OC-48, OC-192, or even OC-768 time division multiplexing (TDM) speed.

Table IV summarizes the restoration time in milliseconds for each of these networks under the worst failure scenario. We also count the maximum number of cross-connect activations at a single node. This maximum cross-connect load is actually the bottleneck for restoration speedup, because cross-connect request messages must be processed sequentially at each node, even though cross-connect execution can be done in parallel.

These results show that subsecond restoration is achievable even for very large networks carrying nationwide traffic. As mentioned earlier, much faster restoration is possible with faster processors in cross connects.

Optimization Performance

We now evaluate the performance of our restoration algorithms in terms of capacity utilization optimization. The distributed algorithm needs to be evaluated in a different way from the centralized algorithms because of its stochastic element.

The distributed algorithm. Ideally, the distributed algorithm should be evaluated by a complete simulation model that can simulate the entire process of the algorithm. Unfortunately, such a model would be extremely complex and slow to run for a realistic length of simulation time. As an alternative, a simpler simulation model is used to capture only the dynamics of demand contention locking. By assuming a route search time for every demand once it successfully locks out its contenders, the simulation model can determine the order in which demands conduct their path searches. Once the demand ordering is given, a deterministic procedure can take over the computation from this point forward. Since noncontending demands can do their path searches in parallel, this ordering does not necessarily result in a single sequence. However, this parallelism does not add any complexity to the deterministic procedure because there is no conflict between noncontending demands in their capacity consumption.

The optimization performance of the distributed algorithm is measured by how many restoration routes it can find compared with those of the optimum solution, given fixed link capacities. We choose the network A introduced previously as our test example, because a small network makes it easy to find the optimum solution and to make the simulation run. We create 10 test problems out of this network by randomly generating 10 sets of link capacities, with the capacity of each link being in the range of 1 to 5. For each of the 10 problems, **Table V** lists the number of

Table VI. Network and traffic sizes.

Network	Number of nodes	Number of links	Number of demands in set 1	Number of demands in set 2	Number of demands in set 3	Number of demands in set 4	Number of demands in set 5
D	28	47	100	150	200	250	300
E	83	135	200	300	500	700	1,000

Table VII. Comparison of algorithms for network D.

Demand set	Shortest-path pair	Algorithm L	Algorithm G	Lower bound
100	539	461	425	388
150	811	714	667	607
200	1,061	933	874	806
250	1,326	1,190	1,124	1,030
300	1,583	1,420	1,355	1,235

Table VIII. Comparison of algorithms for network E.

Demand set	Shortest-path pair	Algorithm L	Algorithm G	Lower bound
200	1,758	1,575	1,490	1,304
300	2,938	2,630	2,516	2,181
500	5,353	4,881	4,599	3,987
700	7,856	7,256	6,776	5,867
1,000	11,556	10,673	9,882	8,502

demands whose restoration routes could not be found by the algorithm, as compared with those of the optimal solution. The total number of demands is 50.

These limited test results indicate that even a very primitive and distributed algorithm can produce solutions very close to the optimum, most of time. Actually, for these particular test cases, it matches the optimal solutions 5 out of 10 times. Furthermore, note that the optimal solution assumes a centralized algorithm, so it may not be achievable by any distributed algorithm.

Centralized algorithms for problem 2. In evaluating the centralized algorithms, we choose to use two examples: network D and network E, each with multiple demand sets. For each of the networks, we use five different sets of demands (varying in size) for the testing. The network and demand sizes are summarized in **Table VI**. In this case, the optimization performance of an algorithm can be measured by the total capacity required to route all the demands (both service and restoration routes). Due to the size of the problems, we are unable to obtain the exact optimal solutions. We use the shortest disjoint path-pair solution as a reference datapoint. When calculating the capacity requirement by this solution, the shorter path of a path pair is taken as the service route. Then, capacity sharing is calculated among the restoration routes. We view this as a common-sense solution. In addition, we also provide the lower bounds generated by the Lagrangean relaxation approach as another reference datapoint for accessing the optimality of our solutions.

Tables VII and **VIII** summarize the shortest disjoint path-pair solution of Suurballe's algorithm, the results of algorithm G, the results of algorithm L, and the Lagrangean relaxation lower bound for network D and network E, respectively.

The total capacity can be divided into primary (service) capacity and restoration capacity. The primary capacity is the capacity consumed by the service routes, and the restoration capacity is that consumed by the restoration routes. There are two ways to calcu-

Table IX. Capacity breakdown by algorithm L on network D.

Demand set	Primary capacity	Restoration capacity with sharing	Restoration capacity without sharing	Total capacity with sharing
100	262	199	421	461
150	407	307	649	714
200	543	390	870	933
250	692	498	1,083	1,190
300	835	585	1,306	1,420

Table X. Capacity breakdown by algorithm L on network E.

Demand set	Primary capacity	Restoration capacity with sharing	Restoration capacity without sharing	Total capacity with sharing
200	922	653	1,866	1,575
300	1,534	1,096	3,067	2,630
500	2,758	2,123	5,580	4,881
700	3,963	3,293	8,027	7,256
1,000	5,775	4,898	11,750	10,673

late restoration capacity. One is to consider capacity sharing among restoration routes. Another is to assume that each demand has dedicated capacity reserved on its restoration route (as in the case of path-switched SONET/SDH rings). This division helps us to gain insight on network capacity overbuild for restoration and on the impact of restoration capacity sharing. Restoration without restoration capacity sharing is equivalent to 1 + 1 restoration. If capacity overbuild is of no concern, 1 + 1 restoration offers the fastest possible restoration and the simplest possible implementation architecture. Even when the network capacity is not abundant for 1 + 1 protection everywhere, it may be desirable to partition demands into different priority traffic classes. One may want to use capacity-dedicated restoration routes for traffic with higher priority and capacity-shared restoration routes to protect traffic with lower priority.

Tables IX and **X** report more detailed results of algorithm L (summaries of primary capacity, restoration capacity, capacity sharing, and no capacity sharing) for networks D and E, respectively.

We would like to make some observations from these results. First, from a capacity optimization point of view, the solutions of algorithm G are approximately 20% better than those of the shortest path pair and are within 10% deviation from the lower bounds. This indicates that our optimization approach makes a significant difference from the common-sense design. It also shows that algorithm G produces very optimal solutions. A 10% deviation from the lower bound implies an even smaller gap with the true optimum. The comparison also shows that algorithm L underperforms algorithm G only by about 7% on average, but at much less computation complexity. Second, restoration capacity overbuild with restoration capacity sharing is between 60% and 90% for mesh networks with our optimization algorithms. The overbuild for the shortest disjoint path-pair solution with capacity sharing is usually about 100% (not reported in this paper). Third, the capacity sharing between restoration routes has a significant impact on the capacity requirement. Without capacity sharing, the overbuild approaches 200%. Surely, the difference between capacity sharing and no sharing is significant for the solutions of our algorithms because of the optimization. However, even for the shortest path pair solution, overbuild without capacity sharing is usually more than 150%. Note, however, that the capacity overbuild might be a misleading indicator if the service routes are not selected according to common sense. If the primary routing is shortest path based, the restoration capacity overbuild will be a percentage of the most efficient primary capacity. In that case, 100% overbuild does not seem unacceptable. On the other hand, if the service routes are unnecessarily long, the restoration capacity overbuild will be a fraction of this excessive primary capacity. In this case, even a 50% overbuild can be a very inefficient one. A fair comparison of capacity efficiency between two different restoration schemes is either by capacity overbuild with common primary routing or by total capacity requirement.

Scalability and Application to Service Layer Restoration

While the traffic matrix used in the examples above represents high volume in terms of OC-48 or OC-192 traffic per wavelength, the expected explosion in data traffic may increase the volume significantly. Thus, it is important to ensure that the algorithms for precomputation of restoration routes and execution of restoration are scalable, as the number of wavelengths to be routed increases. Furthermore, as we mentioned earlier, the same approaches are applicable to provision and restore demands at higher layers (for example, SONET/SDH, ATM, and IP). In these cases, the demands (SONET/SDH circuits, virtual paths, and sets of flows) may be at a much lower granularity than the bandwidth of full wavelength. Once again, the number of demands will be much larger, and scalability will become important.

We note that the number of desirable routes between a given pair of nodes depends on the topology of the network and not on the number of demands. Typically, this number of desirable routes will be small. Thus, as the number of demands grows, more and more demands between the same pair of nodes are likely to be routed along the same route. A simple approach to scalability is to group the demands between a pair of nodes into groups of size $\leq K$ and to consider each group as a single demand for routing purposes. By making K proportional to the total number of demands in the whole network, the algorithms scale readily as the number of actual demands grows. While some flexibility in routing is lost, the impact on optimality is minimal.

By the discussion above, we may assume that the total number of demands is a function of n, where n is the number of nodes, and that its value lies between cn and n^2, where c is a constant. For the distributed algorithms, in a conceptual sense, the computational complexity at each node is proportional to the demand density—that is, the average number of demand terminations at a single node or the average number of demands carried by a single link (by shortest-path routing, for example). We argue that, while the number of demands can grow superlinearly as a function of n, the demand density is likely to be a constant or, at most, a linear function of n. Therefore, the scalability of our distributed algorithms can be estimated by inflating the number of nodes. Our test results indicate that the algorithms run very fast with networks up to 500 nodes.

Conclusion

In this paper, we presented design and restoration algorithms for optical mesh networks. The restoration algorithm is completely distributed and capable of achieving subsecond restoration under realistic application scenarios. Its failure independence, restoration architecture, contention-locking mechanism, and restoration capacity sharing scheme make the algorithm robust, fast, and capacity efficient. Our preliminary test and simulation results support these observations. Work is in progress to design a faster detection and activation architecture to bring the restoration time down below 100 ms.

We also considered the joint design and restoration problem. We presented two algorithms that focus on optimizing capacity utilization within a reasonable computational complexity. We benchmarked the optimization performance of these algorithms with that of the shortest disjoint path solution in two test cases representing large nationwide networks. Our results offer some important insights on restoration capacity overbuild, on the difference between capacity sharing and capacity dedicating, and on the tradeoff between restoration speed and capacity efficiency. One of the algorithms we presented can easily be converted into a distributed algorithm. This algorithm and its test results will be presented in a future paper.

The algorithms we presented in this paper are applicable to SONET/SDH mesh restoration as well as service layer (ATM, IP) restoration. Demand bundling will allow the algorithms to scale in these situations.

Appendix A. Optimization Procedure

First, we need to define terms and introduce notation. Let d be a demand that does not find a restoration route in the search procedure described above and, therefore, activates the optimization procedure. Let s_d and t_d be d's source node and destination node, respectively. Since the BFS started at s_d and failed to reach t_d, it partitions the set of nodes into two subsets V_d and W_d, where V_d is the set of nodes that are reachable from s_d via valid routes and W_d is its complement. Following network flow terminology, we define a cut C_d as the set of links between V_d and W_d (that is, one end in V_d and the other in W_d). Obviously, any path from s_d to t_d must traverse at least one of the links in C_d and, currently, none of these links has capacity for d. For each link $l \in C_d$, let Q_l be the set of demands that are d's critical contenders over l, as determined by the link capacity control procedure. That is, path removal of any one of the demands in Q_l from l would release capacity for d. Let $B_d = \bigcup_{l \in C_d} Q_l$, known as the set of cut contenders of d. The basic idea of the optimization algorithm is to reroute (change) the restoration route of one of d's cut contenders to release some critical links for d. The following is an outline of the algorithm:

1. For each $k \in B_d$, take out the restoration route A_k of k and release capacity for all the links on the route.
2. Execute the route search procedure for d. If no route is found, restore route A_k for k, go to step 1, and continue the loop with a different k. Otherwise, let A_d be the route generated. Update capacity tables for all the links on A_d. Go to the next step.
3. Execute the search procedure for the restoration route for k. If one is found, the procedure is finished. We have found restoration routes for both d and k. Break the loop and stop. Otherwise, calculate the cut C_k as a byproduct of the route search for k and go to the next step.
4. Reroute A_d: Take out A_d and release capacity for all the links on the route. For each link in the network that has capacity for d, assign a unit cost, except for links in C_k, which are assigned a high cost. Run a shortest-path algorithm to find a minimum cost route A_d'. Update capacity for all the links on A_d'.
5. Execute the route search procedure again for k. If one is found, the procedure is finished. Break the loop and stop. Otherwise, restore route A_k for k, go to step 1, and continue the loop with a different k.

Note that a more sophisticated locking mechanism is needed in order to implement the optimization algorithm in a distributed way. When demand d is negotiating with demand k, other demands contending to either of them have to be locked out. This can easily be done by modifying the basic contention-locking procedure to lock both service routes.

Appendix B. Proof of the Theorem

We now provide a proof of the theorem regarding the relationship between (IP$_d$) and (IP).

Theorem. Let v_d be the optimum value of (IP$_d$) for each $d \in D$, and let v be the optimum value of (IP). Then, $\sum_{d \in D} v_d \leq v$.

Proof. Let $\left\{ \overline{X}_{ij}^d, \overline{Y}_{ij}^d, \overline{Z}_{ijf}^d, \overline{W}_{ij} \right\}$ be an optimum solution of problem (IP). For each demand $d \in D$, let P_d and A_d be the primary path and alternate path determined by $\left\{ \overline{X}_{ij}^d \right\}$ and $\left\{ \overline{Y}_{ij}^d \right\}$, respectively. Suppose P_d and A_d are not disjoint—for example, suppose they share fault f. Then under fault f, either constraint (4) or constraint (5) will be violated. Thus, P_d and A_d are disjoint. Therefore, they are a feasible solution of problem (IP$_d$). Now,

$$v = \sum_{(ij) \in L} C_{ij} \overline{W}_{ij} \geq \sum_{(ij) \in L} C_{ij} \max_{f \in F} \left(\sum_{d \in D} \overline{X}_{ij}^d + \sum_{d \in D} \overline{Z}_{ijf}^d \right)$$

$$= \sum_{(ij) \in L} C_{ij} \sum_{d \in D} \overline{X}_{ij}^d + \sum_{(ij) \in L} C_{ij} \max_{f \in F} \sum_{d \in D} \overline{Z}_{ijf}^d$$

$$= \sum_{d \in D} \sum_{(ij) \in L} C_{ij} \overline{X}_{ij}^d + \sum_{(ij) \in L} C_{ij} \max_{f \in F} \sum_{d \in D} \overline{Z}_{ijf}^d .$$

Note that $\sum_{d \in D} \overline{Z}_{ijf}^d = \sum_{d \in D:\, f \in P_d, (ij) \in A_d} 1$ and $\sum_{f \in F, (ij) \in L_f} \lambda_{ijf} = C_{ij}$.

Thus,

$$v \geq \sum_{d \in D} \sum_{(ij) \in L} C_{ij} \overline{X}_{ij}^d + \sum_{(ij) \in L} \sum_{f \in F:(ij) \in L_f} \lambda_{ijf} \max_{f \in F} \sum_{d \in D:\, f \in P_d, (ij) \in A_d} 1$$

$$\geq \sum_{d \in D} \sum_{(ij) \in L} C_{ij} \overline{X}_{ij}^d + \sum_{(ij) \in L} \sum_{f \in F:(ij) \in L_f} (\lambda_{ijf} \sum_{d \in D:\, f \in P_d, (ij) \in A_d} 1)$$

$$= \sum_{d \in D} \sum_{(ij) \in L} C_{ij} \overline{X}_{ij}^d + \sum_{d \in D} \sum_{(ij) \in A_d} \sum_{f \in P_d} \lambda_{ijf}$$

$$= \sum_{d \in D} \sum_{(ij) \in P_d} C_{ij} + \sum_{d \in D} \sum_{(ij) \in A_d} \sum_{f \in P_d} \lambda_{ijf}$$

$$= \sum_{d \in D} \left(\sum_{(ij) \in P_d} C_{ij} + \sum_{(ij) \in A_d} \sum_{f \in P_d} \lambda_{ijf} \right) .$$

Since P_d and A_d are a feasible solution to (IP$_d$), $\sum_{(ij) \in P_d} C_{ij} + \sum_{(ij) \in A_d} \sum_{f \in F_{pd}} \lambda_{ijf} \geq v_d$. Therefore, $v \geq \sum_{d \in D} v_d$.

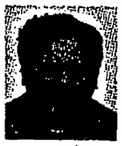
References

1. T.-H. Wu, *Fiber Network Service Survivability*, Artech House, Norwood, Mass., 1992.
2. W. D. Grover, "The Self-Healing Network: A Fast Distributed Restoration Technique for Networks Using Digital Cross-Connect Machines," *Proc. IEEE GLOBECOM '87*, Tokyo, Nov. 1987, pp. 1090–1095.
3. C. H. Yang and S. Hasegawa, "FITNESS: Failure Immunization Technology for Network Services Survivability," *Proc. IEEE GLOBECOM '88*, Hollywood, Fla., Nov. 1988, pp. 1549–1554.
4. R. Kawamura, K. Sato, and I. Tokizawa, "Self-Healing ATM Networks Based on Virtual Path Concept," *IEEE JSAC Special Issue: Integrity of Public Commun. Networks*, Vol 12, No 1, 1994, pp. 120–127.
5. K. Struyve, P. Demeester, L. Nederlof, and L. Van Hauwermeiren, "Design of Distributed Restoration Algorithms for ATM Meshed Networks," *Proc. IEEE 3rd Symp. on Commun. and Vehicular Technology*, Eindhoven, Netherlands, Oct. 1995, pp. 128–135.
6. K. Murakami and H.S. Kim, "Optimal Capacity and Flow Assignment for Self-Healing ATM Networks Based on Line and End-to-End Restoration," *IEEE/ACM Trans. on Networking*, Vol. 6, No. 2, 1998, pp. 207–221.
7. J. Anderson, B. Doshi, S. Dravida, and P. Harshavaradhana, "Fast Restoration of ATM Networks," *IEEE JSAC*, Vol. 12, Jan. 1994, pp.128–138.
8. J. Carlier and D. Nace, "A New Distributed Restoration Algorithm for Telecommunication Networks," *Proc. NETWORKS '94*, 1994, pp. 377–380.
9. B. A. Coan, W. E. Leland, M. P. Vecchi, A. Weinrib, and L. T. Wu, "Using Distributed Topology Update and Preplanned Configurations to Achieve Trunk Network Survivability," *IEEE Trans. on Reliability*, Vol. 40, No. 4, Oct. 1991, pp. 404–416.
10. R. R. Iraschko, W. D. Grover, and M. H. MacGregor, "A Distributed Real-Time Path Restoration Protocol with Performance Close to Centralized Multi-Commodity Maxflow," *Proc. 1st Intl.*

Workshop on Design of Reliable Commun. Networks, Brugge, Belgium, May 1998, paper O-9.
11. C.-W. Chao, P. M. Dollard, J. E. Weythman, L. T. Nguyen, and H. Eslambolchi, "FASTAR: A Robust System for Fast DS3 Restoration," *Proc. IEEE GLOBECOM '91,* Phoenix, Ariz., Dec. 1991, pp. 1396–1400.
12. D. K. Doherty, W. D. Hutcheson, and K. K. Raychaudhuri, "High-Capacity Digital Network Management and Control," *Proc. IEEE GLOBECOM '90,* San Diego, Calif., Dec. 1990, pp. 60–64.
13. R. D. Doverspike and C. D. Pack, "Using SONET for Dynamic Bandwidth Management in Local Exchange Networks," *NETWORKS '92,* Birmingham, England, 1992.
14. C. Palmer and F. Hummel, "Restoration in a Partitioned Multibandwidth Cross-Connect Network," *Proc. IEEE GLOBECOM '90,* San Diego, Calif., Dec. 1990, pp. 81–85.
15. C. E. Chow, S. McCaughey, and S. Syed, "Rreact: A Distributed Protocol for Rapid Restoration of Active Communication Trunks," *UCCS Tech Report EAS-CS-92-18,* Nov. 1992.
16. C. E. Chow, J. Bicknell, S. McCaughey, and S. Syed, "A Fast Distributed Network Restoration Algorithm," *Proc. 12th Intl. Phoenix Conf. on Computers and Commun.,* Tempe, Ariz., Mar. 1993, pp. 261–267.
17. H. Komine, T. Chujo, T. Ogura, K. Miyazaki, and T. Soejima, "A Distributed Restoration Algorithm for Multiple-Link and Node Failures of Transport Networks," *Proc. IEEE GLOBECOM '90,* San Diego, Calif., Dec. 1990, pp. 459–463.
18. R. Kawamura and I. Tokizawa, "Self-Healing Virtual Path Architecture in ATM Networks," *IEEE Commun. Magazine,* Vol. 33, No. 9, Sept. 1995, pp. 72–79.
19. A. Balakrishnan, T. Magnanti, J. Sokol, and Y. Wang, "Modeling and Solving the Single Facility Line-Restoration Problem," Technical Report, MIT Operations Research Center, Apr. 1998.
20. W. D. Grover, T. D. Bilodeau, and B. D. Venables, "Near Optimal Spare Capacity Planning in a Mesh Restorable Network," *Proc. IEEE GLOBE-COM '91,* Phoenix, Ariz., Dec. 1991, pp. 2007–2012.
21. H. Sakauchi, Y. Nishimura, and S. Hasegawa, "A Self-Healing Network with an Economical Spare-Channel Assignment," *Proc. IEEE GLOBE-COM '90,* San Diego, Calif., Dec. 1990, pp. 438–443.
22. A. Lisser, R. Sarkissian, and J. Vial, "Optimal Joint Synthesis of Base and Spare Telecommunication Networks," *Intl. Symp. on Mathematical Programming,* Lausanne, Switzerland, Aug. 1997.
23. M. Stoer and G. Dahl, "A Polyhedral Approach to Multicommodity Survivable Network Design," *Numerische Mathematik,* Vol. 68, No. 1, 1994, pp. 149–167.
24. S. B. Alexander, R. S. Bondurant, D. Byrne, V. W. S. Chan, S. G. Finn, R. Gallagher, B. S. Glance, H. A. Haus, P. Humblet, R. Jain, I. P. Kaminow, M. Karol, R. S. Kennedy, A. Kirby, H. Q. Le, A. A. M. Saleh, B. A. Schofield, J. H. Shapiro, N. K. Shankaranarayanan, R. E. Thomas, R. C. Williamson, and R. W. Wilson, "A Precompetitive Consortium on Wideband All-Optical Networks," *J. Lightwave Tech.,* Vol. 11, No. 5–6, May–June 1993, pp. 714–735.
25. I. Chlamtac, A. Ganz, and G. Karmi, "Purely Optical Networks for Terabit Communication", *Proc. IEEE INFOCOM '89,* Vol. 3, Ottawa, Canada, Apr. 1989, pp. 887–896.
26. E. Karasan and E. Ayanoglu, "Effects of Wavelength Routing and Selection Algorithms on Wavelength Conversion Gain in WDM Optical Networks," *IEEE/ACM Trans. on Networking,* Vol. 6, 1998, pp 186–196.
27. M. Kovacevic and A. Acampora, "Benefits of Wavelength Translation in All-Optical Clear-Channel Networks," *IEEE JSAC,* Vol. 14, 1996, pp. 868–880.
28. A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan, "Efficient Routing in Optical Networks," *JACM,* Vol. 43, No. 6, Nov. 1996, pp. 973–1001.
29. R. Ramaswami and K. Sivarajan, "Routing and Wavelength Assignment in All-Optical Networks," *IEEE/ACM Trans. on Networking,* Vol. 3, 1995, pp 489–500.
30. Z. Zhang and A. Acampora, "A Heuristic Wavelength Assignment Algorithm for Multihop WDM Networks with Wavelength Routing and Wavelength Reuse," *IEEE/ACM Trans. on Networking,* Vol. 3, 1995, pp 281–288.
31. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman, New York, 1979.
32. M. L. Fisher, "The Lagrangean Relaxation Method for Solving Integer Programming Problems," *Management Science,* Vol. 27, No. 1, 1981, pp. 1–18.
33. J. W. Suurballe, "Disjoint Paths in a Network," *Networks,* Vol. 4, 1974, pp. 125–145.

(Manuscript approved March 1999)

BHARAT T. DOSHI, who holds a Ph.D. in operations research from Cornell University in Ithaca, New York, is head of the Performance Analysis Department at Bell Labs in Holmdel, New Jersey. Dr. Doshi is responsible for protocol designs, performance analysis, traffic management, routing and restoration algorithms, and architecture of next-generation converged networks. Recent work has focused on wireless, HFC, Internet, SONET, WDM, and ATM technologies, as well as on interworking among these technologies. Dr. Doshi, who has authored more than 100 technical papers and submitted 35 patent applications, is a Bell Labs Fellow and also an IEEE Fellow.



SUBRAHMANYAM DRAVIDA was a technical manager in the Performance Analysis Department at Bell Labs in Holmdel, New Jersey, when this work was performed. He holds a B.Tech. degree in electrical engineering from the Indian Institute of Technology in Madras, as well as M.S.E.E. and Ph.D. degrees in electrical engineering from the Rensselaer Polytechnic Institute in Troy, New York. While at Bell Labs, Dr. Dravida worked on protocols and architectures for wireless networks, as well as protocols for cable networks. His responsibilities included development of algorithms for the design of ATM, SDH, and wireless networks.

P. HARSHAVARDHANA is a technical manager in the Performance Analysis Department at Bell Labs in Holmdel, New Jersey. He holds a B.Tech. degree in electronics from the Indian Institute of Technology in Madras, as well as M.S. and Ph.D. degrees in electrical engineering from the University of Southern California in Los Angeles. Dr. Harshavardhana is working on the design and analysis of optical, SONET/SDH, and data networks. He is responsible for analyzing the network impact of product features and alternatives and for modeling the performance of systems and networks. He has authored numerous technical papers and holds over 20 patents.

ODED HAUSER, a member of technical staff in the Performance Analysis Department at Bell Labs in Holmdel, New Jersey, is currently working on network design, modeling, restoration, and traffic engineering. He holds a B.Sc. degree in information systems and operations research from Technion, Israel Institute of Technology, in Haifa.

YUFEI WANG is a member of technical staff in the Performance Analysis Department at Bell Labs in Holmdel, New Jersey. He holds a B.S. degree in mathematics from the University of Science and Technology of China in Hefei, Anhui, as well as M.S. and Ph.D. degrees in operations research from Cornell University in Ithaca, New York. Dr. Wang is working on network design, modeling, restoration, and traffic engineering. ◆